DOI:10.7544/issn1000-1239.2015.20140268
52(2):318-332,2015

分布式流处理技术综述

崔星灿　禹晓辉　刘　洋　吕朝阳
(山东大学计算机科学与技术学院　济南　250101)
(xccui@mail.sdu.edu.cn)

Distributed Stream Processing: A Survey

Cui Xingcan, Yu Xiaohui, Liu Yang, and Lü Zhaoyang
(*School of Computer Science and Technology, Shandong University, Jinan* 250101)

Abstract　The rapid growth of computing and networking technologies, along with the increasingly richer ways of data acquisition, has brought forth a large array of applications that require real-time processing of massive data with high velocity. As the processing of such data often exceeds the capacity of existing technologies, there has appeared a class of approaches following the distributed stream processing paradigm. In this survey, we first review the application background of distributed stream processing and discuss how the technology has evolved to its current form. We then contrast it with other big data processing technologies to help the readers better understand the characteristics of distributed stream processing. We provide an in-depth discussion of the main issues involved in distributed stream processing, such as data models, system models, storage management, semantic guarantees, load control, and fault tolerance, pointing out the pros and cons of existing solutions. This is followed by a systematic comparison of several popular distributed stream processing platforms including S4, Storm, Spark Streaming, etc. Finally, we present a few typical applications of distributed stream processing and discuss possible directions for future research in this area.

Key words　big data; data stream; distributed stream processing; real-time processing; distributed system

摘　要　随着计算机和网络技术的迅猛发展以及数据获取手段的不断丰富,在越来越多的领域出现了对海量、高速数据进行实时处理的需求.由于此类需求往往超出传统数据处理技术的能力,分布式流处理模式应运而生.首先回顾分布式流处理技术产生的背景以及技术演进过程,然后将其与其他相关大数据处理技术进行对比,以界定分布式流数据处理的外延.进而对分布式流处理所需要考虑的数据模型、系统模型、存储管理、语义保障、负载控制、系统容错等主要问题进行深入分析,指出现有解决方案的优势和不足.随后,介绍 S4,Storm,Spark Streaming 等几种具有代表性的分布式流处理系统,并对它们进行系统地对比.最后,给出分布式流处理在社交媒体处理等领域的几种典型应用,并探讨分布式流处理领域进一步的研究方向.

关键词　大数据;数据流;分布式流处理;实时处理;分布式系统

中图法分类号　TP391

收稿日期:2014-11-18;**修回日期**:2014-12-10

基金项目:国家自然科学基金项目(61272092);山东省自然科学基金项目(ZR2012FZ004);山东省科技发展计划基金项目(2014GGE27178);国家"九七三"重点基础研究发展计划基金项目(2015CB352500);山东大学自主创新基金项目(2012ZD012);泰山学者计划基金项目

通信作者:刘　洋(yliu@sdu.edu.cn)

随着计算机技术和网络技术的发展,在越来越多的行业领域产生了以“海量”和“高速”为特征的数据.这里,“海量”是指数据规模大,“高速”是指数据产生、到达的速率高.在过去几年中,对这些海量高速数据进行实时处理和分析的应用需求正呈爆炸式增长.例如,在工程及运输领域,实时地对工作部件在运行过程中快速产生的大量数据进行处理和分析,可以及时了解部件的当前工作状态,在异常产生时立即发出报警信号,有利于对作业环境中风险的控制.一个典型的例子是波音 737 飞机.在飞行状态下,737 飞机的引擎每小时将会产生高达 20 TB 的数据,这些数据为了解引擎运行状态提供了一个全方位视角,如能对这些数据进行实时分析,可以推断出目前引擎工作是否正常,从而为飞行安全提供强有力的保障.

又如在移动通信领域,世界范围内每年由电信诈骗行为造成的损失约为 400 亿美元,很大程度上是由于发现诈骗行为的滞后性.究其原因,主要是由于电信数据种类繁多、数量巨大,在一个大型城市中,平均每分钟会产生超过 8 万条位置更新数据,每天网络承载流量可高达 100 TB.若能对这些海量数据进行实时挖掘分析,便可较为准确地识别出涉及电信诈骗的通信请求,降低由于发现滞后引起的损失.

再如,据路透中文网报道,在 2014 年“双十一”期间,支付宝平台最多每分钟收到 285 万笔交易信息,阿里内部的数据分析人员需要实时对这些交易数据完成风险检测、数据校验、聚合统计、可视化展示等工作.

总的来说,上述应用场景具有以下共同特点:

1) 数据规模庞大,往往达到 PB 级别;数据产生速度极快,可以达到 GB/s 级别;

2) 处理过程往往会涉及到复杂的数学模型,需要提供强力的底层支持,以保证这些模型在海量、高速数据环境中可以高效工作;

3) 数据的时效性很强,导致对数据处理过程的整体延迟要求非常苛刻,必须在秒级或更短的时间内得到结果,从而进一步作出反应.

这些与传统应用场景不同的特点,使得现有的技术不能很好地满足对海量高速数据进行实时处理和分析的需求.即使是 HDFS[1]/MapReduce[2] 这种近年来被业界广泛采用的海量数据处理架构,也并不适用于如此高速和复杂的实时数据处理和分析场景中.原因在于 HDFS/MapReduce 主要是面向对静态数据的批处理,其使用外存作为中间结果数据的存储介质,巨大的 I/O 代价会成为影响处理过程实时性的瓶颈.

鉴于此,海量高速数据的实时处理已经引发了越来越多的关注,一些新的技术已经萌芽.其中,一类面向快速、不断产生的数据进行处理、并立即产生结果的流处理(stream processing)模式得到了迅猛发展.由于面对的是不停更新的动态数据,流处理系统的延迟通常很低,因此该系统常常被应用于构建实时系统(real-time system).

从历史发展来看,流处理技术并非大数据时代的产物.早在 20 世纪 90 年代,它便被应用到报警环境中[3].期间,实时数据库(real-time database)[4]、主动数据库(active database)[5]、信息过滤系统(information filtering system)[6]等技术作为流处理早期形态,在各自领域都有发展.

20 世纪初,数据流管理系统(data stream management system)开始受到重视,出现了包括 Aurora[7],TelegraphCQ[8],STREAM[9] 等在内的各具特色的数据流管理系统.这些早期系统和本文所要重点介绍的分布式流处理系统(distributed stream processing system)在目标数据、处理模式、系统要求方面有着很多相同或相似之处,因而诸多研究工作也互有交叉.相比分布式流处理系统,早期的数据流管理系统具有以下 4 个特点:

1) 应用领域较为单一,通常是针对某个特定场景,且多数不需要保障对每一条数据的精确处理;

2) 多数采用集中式架构,很少涉及一致性、容错、高可用性等复杂问题;

3) 提供基本算子和查询语言,允许用户方便地在数据流上完成查询和处理操作;

4) 算子实现和系统底层模块耦合度较高,难以进行扩展开发.

在一些特殊应用场景下,流数据源天生的分散性质和数据量的不断增长使人们开始意识到分布式技术对于流处理的重要性,Medusa[10],Flux[11],Borealis[12] 等分布式数据流管理系统相关技术开始在集中式系统的基础上有所发展.

2004 年以来,随着 Hadoop[13] 平台的诞生,各类开放式计算平台逐渐兴起.类似于分布式 NoSQL 数据库之于传统关系型数据库,分布式流处理技术已接替集中式流处理技术,成为大数据时代的焦点,S4[14],Storm[15],SparkStreaming[16] 等面向流处理

的平台相继被提出. 和以往的数据流管理系统不同①,这些平台都采用分布式架构,其处理能力可以随节点数目的增长而扩展,具有良好的伸缩性(scalable). 此外,多数平台都将计算逻辑和基础模块分离,自身只完成底层数据传输、任务分配等工作,并不提供查询语言支持,用户需要以编码方式自行完成处理流程和计算单元的定义. 总结起来,导致流处理系统平台化的主要原因有两点:

1) 和集中式架构相比,分布式架构在数据模型、负载控制、语义保障、高可用性等方面,给底层模块的设计提出了更高的挑战,系统平台化后基础模块与计算逻辑分离,可以专注于研究独立问题的通用解决方案;

2) 随着数据种类及应用场景的丰富,传统流处理系统所提供的算子及查询语言难以满足用户对实时数据处理复杂度的要求,如处理非结构化数据、持续更新外部状态等,开放的平台化设计有利于适应多样化的数据处理需求.

20 世纪 10 年代,一类和流处理相关的研究——复杂事件处理(complex event processing)——也得到了较快发展. 从概念上看,此类研究关注点更加集中,它们针对各类事件流数据,侧重于如何从事件中发现某些复杂模式(complex pattern),从而可以作出相应反应. 和复杂事件处理相关的研究有 SASE[17-18], Cayuga[19-21]等.

面对大数据海量(volume)、高速(velocity)及多样(variety)的特点,分布式流处理系统以其伸缩性、实时性和开放性应对数据处理带来的一系列挑战,在大数据时代必将具有举足轻重的地位. 文献[22]介绍了大数据流处理技术和相关实例,但重点放在实例分析上. 本文将对分布式流处理相关技术进行较为全面、系统的剖析.

1 大数据并行处理模式比较

近些年,随着数据规模和复杂程度的提升,大数据开始引起社会各界的高度关注[23]. 为了从这些不断产生的数据中提取有价值的信息,并更好地处理、利用这些资源,学术界和工业界都提出了很多包括分布式流处理在内的、面向大规模数据的并行处理模式(parallel paradigm)[24]. 由于数据规模不断增大,这些模式基本上都采用分布式技术来提升计算或系统本身的可伸缩性. 从数据形态、处理粒度②及依托介质 3 个维度进行考虑,可以将这些处理模式粗略概括为 5 类,如图 1 所示:

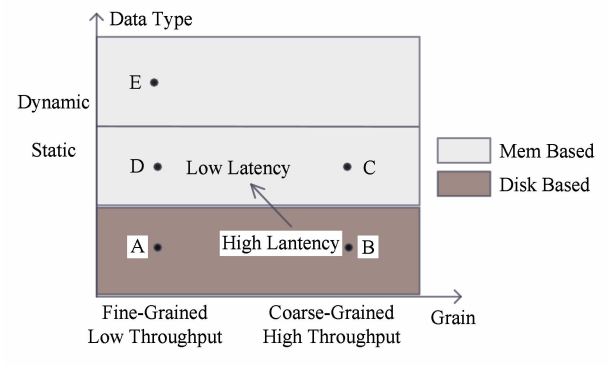


Fig. 1 Parallel data processing paradigms.

图 1 并行数据处理模式

A 类是面向基于磁盘静态数据的细粒度处理模式,其代表方案是利用分布式数据库作为数据来源,进行传统的数据获取、处理操作. 此类模式比较适合计算复杂、对吞吐量要求不高同时需要快速响应的应用场景.

B 类是面向基于磁盘的静态数据的粗粒度处理模式,其代表是 MapReduce,同 A 类相比,此类模式拥有更高的吞吐量,但数据处理延迟也会随之增加,比较适合完成对响应要求不高的离线批处理任务.

C,D,E 三类处理模式都是基于分布式内存,可看作是内存计算(in-memory computing)的具体实现. 其中 C 类主要是面向基于内存的静态数据进行粗粒度计算,一个典型的代表是 Berkeley 提出的 Spark[25]系统. 相比 A,B 两类基于磁盘的处理模式而言,C 类模式的数据处理速度可以高出 1～2 个数量级;但由于内存容量限制和数据易失性等特点,该模式也为整体解决方案的设计提出了更高的要求.

D 类处理模式和 A 类相似,区别在于把基于磁盘的分布式数据存储替换为基于内存,如基于分布式内存数据库和分布式键值存储(distributed key-value store)的计算. 其局限性与 C 类模式相似.

E 类面向动态数据的细粒度处理模式,其核心技术之一即是分布式流处理. 同其他几类处理模式相比,此类处理模式基于不断产生的动态数据,拥有最低的处理延迟,因此非常适合监控系统、在线金融分析、算法交易等对实时性要求较高的应用场景. 由

① 下文将以流处理系统代指数据流管理系统和流处理平台

② 指单次操作所面向的最小数据单元

于需要处理的目标数据集无限且随时产生,因此只要不在高峰时期产生数据堆积,系统对于吞吐量并无过高要求.同时,由于处理粒度很细(经常是单条记录),因而系统在故障恢复(fault recovery)方面,较其他粗粒度数据处理系统将会面临更大挑战[11].

2 分布式流处理系统相关问题

设计和实现一个完善的分布式流处理系统通常有很多需要考虑的问题,如数据抽象、负载均衡、故障恢复等.我们将其中一些较为关键的问题进行如图2所示的分类①,下文将会分别对问题本身及已有解决方案进行分析和回顾.值得注意的是,不同类别问题之间有时会存在紧密联系,例如系统所提供的存储管理机制将会直接影响其故障恢复的策略选择,我们会将这些联系在相关问题中进行阐述.

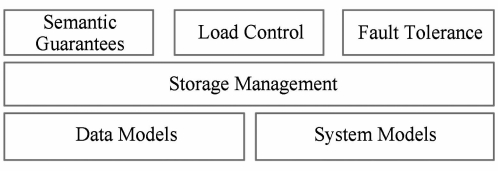


Fig. 2 Main problems for distributed stream processing system.

图2 分布式流处理系统问题分类

2.1 数据模型

无论是集中式还是分布式的流处理系统,如何对处理数据进行抽象建模都是一个首先要明确的问题.直觉上,数据流应该是某种数据项连续不断的序列,单个数据项既可以看作是关系数据库中的一个元组,也可以看成某种对象的一个实例[26],下文中我们会把这样的数据项统称为记录(record).

由于聚合等操作需要全局数据才能获得结果,因此一个普遍的做法是对数据流进行切分②,对局部的有限数据进行计算,切分的常用方法有界标模型和窗口模型[27-28].即使将无限的数据流分割成有限部分,由于集中式架构在内存和计算能力方面的局限,很多较为复杂的查询或数据挖掘算法依然难以实施.为此,研究人员针对不同种类的数据,提出了一系列基于概述(sketch)、直方图(histogram)、抽样(sampling)等手段的数据概要方法[29],以牺牲准确度为代价换取可接受的空间和计算复杂度,变相缓解了这一问题.后来随着硬件及分布式技术的发展,内存瓶颈已基本解除,数据概要方法的重要性因而逐步降低,当前大多数系统都是基于全部原始数据进行处理.

2.1.1 批次模型

通常,流处理系统中数据会以单条记录的形式传播和处理,如S4平台中的event.分布式系统的两个特征对流处理数据模型产生了一定的影响:1)数据需要通过网络在节点之间进行传输,在某些特定场景下,单次传输较小的数据项会对处理效率产生影响;2)分布式系统节点故障概率增大,对于故障恢复的要求也有所提高,单条记录处理给恢复工作带来更多额外负担[11].鉴于上述两个原因,大多数分布式流处理平台都会支持物理传输层面的批次模型,而Storm,MillWheel[30]等平台更是通过在逻辑层面引入批次概念来实现"事务性"处理(详见2.4节所述).

批次模型的引入一定程度上会增加数据处理延迟,但都没有对流处理的计算粒度产生影响.Spark Streaming的出现则改变了这种细粒度的处理模式.同Spark框架类似,Spark Streaming也是基于弹性分布式数据集(resilient distributed dataset, RDD)模型[31],它所主张的微批次(mini-batch)数据抽象方式,是将传统模型中单记录的处理粒度粗化为集合的处理粒度,即只允许通过预先定义的操作符对RDD进行可重现的变换(transformation).此举的可取之处在于:1)将批处理和流处理的处理接口统一;2)可以使用作用于RDD上的转换操作方便开发;3)能通过世系(lineage)信息和源数据进行数据重构,减轻了故障恢复代价.事实上,Spark Streaming的数据处理模式已经略微偏离了第1节中关于分布式流处理的界定,是一种介于批处理和流处理之间的产物,但鉴于RDD良好的特性及易用的编程接口,在很多对延迟要求不高且仅限于对数据本身变换处理的情况下,Spark Streaming依旧有着较高的应用价值.

2.1.2 乱序和延迟

很多数据流上的处理方法或查询实现都假设到来数据是按照时间或某种顺序进行排列的.事实上,分布式系统中数据的接收时间并不能代表其生成的时间,而且由于传输原因,也不能保证先生成的数据

① 由于分布式系统天生具有可伸缩性,因此我们不将此问题单独讨论

② 下文中用切分表示对数据流分段,用划分表示并行处理时对数据分组

会先到达,因此这种假设在实际生产环境中很难满足.为了应对数据的乱序和延迟问题,一种解决思路是针对数据处理或查询执行逻辑本身进行修改,使其支持任意顺序到来的数据;另一种思路是从平台层面对乱序数据进行处理,使其对上层操作透明.文献[32]提出一种基于时间窗口的查询处理方案——Window-ID,在定义窗口分隔语义的同时,解决了针对窗口内部乱序数据操作符的实现问题;文献[33]对数据的顺序处理(in-order processing)和乱序处理(out-of-order processing)进行了界定,并通过引入低位线(low watermark)的方式,给出了乱序数据上多个操作原语的实现.MillWheel 平台在继承低位线数据切分方式的同时,以内部计时器(application timer)前进触发业务逻辑,从平台层面较好地解决了数据乱序和延迟问题.在事件流处理领域,文献[34]和文献[35]都给出了针对乱序数据的模式匹配方法.总体来看,多数解决方案都是利用分割策略对部分数据进行重排序,从而得到局部有序的数据模型.

2.2 系统模型

系统模型所涵盖的主要问题是如何实现抽象的流处理逻辑,同时也包括系统整体架构的设计.

绝大多数已有系统都将流处理过程抽象成一个有向图[36].图中的点代表针对数据的操作,有向边则代表数据流向.在集中式数据流管理系统中,查询处理器负责将用户提交的查询实例化为一系列前后相连的基本算子,每个算子负责从上游接收数据,处理后生成新的数据流发往下游.

现有的分布式流处理系统依然沿用了有向图的抽象方式,不同的是算子被替换为允许用户自定义的计算单元.为提高并行度,每个计算单元可能会存在多个实例,它们分布在集群节点之上,负责处理上游数据的不同部分(如图 3 所示).数据按照处理逻辑在节点间以直接(如 socket)或间接(如分布式队列)方式进行传输(详见 2.3 节所述).

集中式流处理系统中的调度器在绝大多数分布式系统中以中心节点的形式体现.作为整个集群的"大脑",中心节点主要负责任务分配和发布、负载调整、集群监控等工作.根据其重要度不同,可以把设有中心节点的分布式系统划分为中心化和弱中心化两类,衡量标准在于弱中心化结构的分布式系统在正常运行后可以脱离中心节点,在系统不出现错误的前提

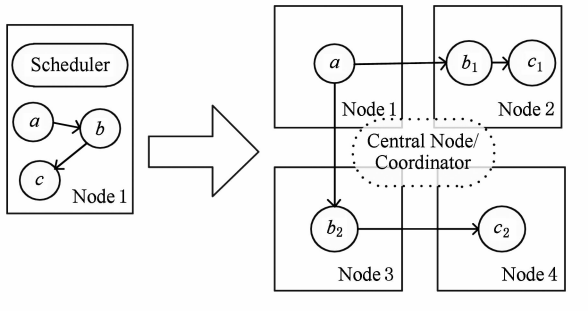


Fig. 3 Distributed computational units.

图 3 计算单元的分布式化

下完成绝大部分工作.如果系统集群中不存在中心节点,完全依靠对等任务节点互相协调,则被称为去中心化结构.S4 就是一个典型的去中心化分布式流处理平台,它遵循 actor 设计模式[37],以数据驱动处理单元(processing element, PE)的创建,理论上每个节点都能独立完成全部数据处理任务.虽然这种去中心化的结构非常符合理想中分布式系统的工作模式,可以有效避免单点失效(single point of failure)问题,但节点之间昂贵的沟通成本往往导致难以实现较为灵活有效的动态负载控制和集群管理策略.

2.3 存储管理

存储管理是指流处理系统对于元数据(meta data)、处理过程中的中间数据(intermediate data)以及全局或局部状态信息(state)的管理和维护.

在集中式流处理系统中,元数据通常会被存储到本地磁盘,上下游处理算子之间以队列方式进行中间数据的传输.由于内存限制,在遇到数据峰值时,算子处理速度落后很可能导致队列溢出.从软件层面考虑,有两种解决方式:1)牺牲效率,启用磁盘缓存更多数据,如 Aurora 通过将数据存入磁盘①,为新数据处理流程的接入和 ad-hoc 查询带来方便;2)牺牲计算精度,按照某种策略舍弃部分数据(详见 2.5 节负载控制中降载的有关内容),如 STREAM 系统.绝大多数集中式流处理系统都把状态当作普通的内存对象存储,没有为其提供特殊的持久化策略.

分布式流处理系统通常会把集群唯一的元数据交由可靠的文件系统或分布式协调工具如 ZooKeeper[38]管理,但对于运行时的状态信息则缺少完美的解决方案.大多数分布式流处理系统的工作节点是以 share-nothing 的形式存在,各自管理维护本地内存中的状态.这种简单方式的不足之处有两点:1)不适用于一些需要全局维护或难以被分散

① Aurora 同样提供减载支持

到各节点的复杂状态结构[39];2)内存状态在节点失效后容易丢失,可能给故障恢复带来较大困难.为应对该问题,Storm 的 Trident① 允许节点状态以增量方式存入远程数据库中;Samza 平台[40]提供本地状态持久化策略并支持远程备份.但这些方案都无法满足高速、可靠的全局存储需求.相关研究工作中 RAMCloud[41],SILT[42]等分布式键值存储在可靠性和故障恢复效率方面还有所欠缺;文献[43]设计并实现了一个全局的基于内存的高可靠性数据存储工具,但提供原语单一,无法满足复杂应用的要求;Tachyon[44]旨在提供一个基于内存的高可靠分布式文件系统,但同样不够完善.

在分布式流处理过程中,中间结果可能需要在节点之间进行传送.和单机总线相比网络的可靠性有所下降,一些系统采用 ZeroMQ[45],Netty[46]等工具来保障数据传输质量,但由于数据暂存于各节点的内存队列或缓冲区中,因此有潜在丢失的危险.为此,MillWheel 提供以 BigTable[47]和 Spanner[48]为支撑的可靠备份仓库(backing store);Spark Streaming 通过世系信息的持久化保证数据可恢复,同时允许中间结果以检查点(checkpoint)方式存储到分布式文件系统;Samza 则是利用分布式队列 Kafka[49]进行数据转发(如图 4 所示),在保障传输质量的同时,也为中间数据提供了一个可靠的存储方式.上述存储方案共同的弊端在于一定程度上降低了数据处理效率,导致延迟增加.

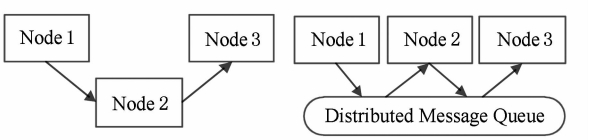


Fig. 4 Forwarding patterns for distributed system.

图 4 分布式系统数据转发模式

综上所述,为满足实时性要求,分布式流处理过程中的数据通常都不应写入磁盘.但从高可用性方面考虑,内存的易失特性往往通过牺牲部分处理效率、引入可靠的持久化存储弥补.

2.4 语义保障

分布式流处理系统中需要对输入数据的处理过程从语义方面进行一定保障.通常按照每条记录被完全处理② 的次数可将处理语义分为 4 类,即无保障、至多处理一次(at most once)、至少处理一次(at least once)以及精确处理一次(exactly once).其中无保障语义对处理无任何约束,因此不作讨论.对其余 3 种处理语义介绍如下:

1) 至多处理一次语义是系统保障每条记录或不被完全处理(中途丢失),或被完全处理一次,不会出现重复现象.该处理语义适用于对数据处理完整性要求不高的场景,例如通过微博内容预测话题趋势,少量微博记录的丢失不会对话题趋势产生太大影响.以推送(push)方式获取数据可以较容易实现至多处理一次的语义.

2) 至少处理一次语义是指系统保障每条记录一定会被完全处理,但可能出现重复处理现象.通常该语义可以在处理路径上每个计算单元都支持幂等操作③ 的场景下使用.为实现该语义,通常需要系统对流入的数据进行持久化存储(如 Spark Streaming 平台),或依赖于可重复获取数据的可靠数据源(如 Kafka 分布式队列),当系统检测到处理失败或超时的情况能够进行数据重发.这种做法本质上是以牺牲效率为代价,克服了流数据"一经处理就很难再次获取的"特性[29].

3) 精确处理一次是所有语义中最严格的一种,它需要系统保障对每个记录完全处理且仅一次,处理后的状态更新往往会反映到可靠的持久化存储介质中.一个典型的应用例子是 WordCount 问题④,对于每条记录中的句子需要精确统计单词频数并进行累加,重复或遗漏都将导致最终结果不准确.为保证该语义,流处理系统需要"记住"哪些数据已经被完全处理.Storm 的 Trident 所采用的一种方法是在保证至少处理一次语义的前提下,将数据流划分为批次,并分别赋予它们唯一的 ID.处理完成后,Trident 负责将对应批次产生的状态更新和 ID 以一次原子操作写入某个可靠的存储介质中.当遇到超时或故障需要进行数据重发时,Trident 会以某种策略生成与之前完全相同的批次数据和 ID,这样如果出现重复提交,可以根据检测 ID 决定是否接受.文献[50]介绍了一种以近似精确处理一次语义⑤ 完成连接(join)操作的方法,其基本实现思路与上述方式相同.

① http://storm.apache.org/documentation/Trident-tutorial.html

② 即单条记录和所有由它产生的新记录已经被途经的全部计算单元所处理

③ 重复相同操作不会对系统产生影响,例如以相同的 key 和 value 向 map 中多次写入

④ 以数据流的方式输入句子,统计每个单词出现的次数

⑤ 近似是因为在某种极端情况下会导致数据丢失

除上述3类语义保障之外,对于一些存在先后顺序(如发生时间)的数据,系统需要在提供精确处理一次语义的前提下,对一些持久化状态相关冲突操作的执行顺序提供一定保障.一些分布式流处理系统借助传统关系数据库中的"事务性"概念① 来帮助解决这一问题.其基本思路是将数据划分为批次,每批次数据所引发的操作和状态更新看作一次"事务"进行.如上文提到的Trident,它采用了一种较为极端的策略,即认为所有批次的状态更新都是存在冲突的,因此只允许每个批次严格按照先后顺序串行执行.MillWheel中冲突处理粒度较Trident更细,它将不同key值的状态分离开,并保证对每一个key值所对应状态的更新会严格按照先后顺序进行.本质上,这些应对措施都是为保证分布式并行处理结果的正确性和一致性.若想精确实现这一目标,需要参照传统关系数据库,首先提供可持久化的原子操作(需要存储策略支持),然后在此基础上对各计算单元实例的冲突依赖关系进行细粒度分析[51],制定并行策略.然而,现有系统的开放性特点以及流数据的动态特性都为其实现带来了一定困难.

2.5 负载控制

负载控制主要是通过一些调度分配策略使得系统可以保持高效稳定运行.在传统集中式架构下,负载控制主要以算子调度(scheduling)和查询优化(query optimization)的形式体现.在分布式系统中,同一类别的计算单元可能存在多个分布在不同节点上的实例,而上游节点的输出数据将直接影响下游节点的任务强度,因此负载控制除了以任务负载均衡(load balancing)的方式体现,还会涉及计算单元的并行度选择(parallelism)、数据划分(partitioning)以及路由策略(routing strategy).无论是集中式还是分布式架构,可以将这些负载控制策略分为两类,即静态策略和动态策略.

1) 静态策略.静态策略是指提前确定、在系统运行过程中无法改变的策略.一种较早针对数据库的静态并行查询策略分为两个步骤[52]:首先,查询优化器根据某个固定的成本模型和历史数据,生成一系列静态的查询计划;然后在执行阶段,这些查询计划中的每一步骤按照某个特定的并发度数被实例化为多个算子实例,分发到不同节点之上.每个算子实例负责执行全部数据的一部分,并且在运行过程中实例数目和所负责的数据子集无法改变.S4平台在数据划分和路由选择方面就是采用了类似的静态策略,无论早期版本通过XML文件进行固定配置,还是新版本指定并行度后将数据按照key值自动划分、路由,一旦策略确定将无法在运行过程中改变.文献[53]中介绍了一个高效静态负载策略的制定细节.

2) 动态策略.在实际生产环境中,由于数据流的无尽性质常常会导致处理任务也会被长期连续执行.不同时间,数据本身的偏向性和运行环境的变化都会对系统负载产生影响,因此静态负载控制策略难以保证总是高效,这时往往需要应用一些允许在运行时改变的动态负载控制策略.集中式系统可以通过运行时调度和动态查询优化的方式来动态调整负载.传统数据流管理系统中,Aurora利用QoS模块反馈为调度提供支持;STREAM使用StreaMon[54]来进行自适应的查询优化;而TelegraphCQ中采用的Eddy[55]则是根据输入数据的特征,在运行时对多元操作符的执行计划进行重新排序,巧妙地降低了系统开销.针对分布式流处理系统,Shah等人把能够给予同类数据划分及路由位置保障的路由策略称为上下文敏感(context sensitive)策略,反之则是随机策略[56].由于涉及聚合等操作,分布式流处理系统都应支持上下文敏感策略.从另一方面考虑,动态策略可分为非自适应和自适应的.前者允许系统在外界干预(例如人工指定)下完成在线的负载均衡等工作,Storm平台就采用了这种策略.早期针对自适应的分布式负载控制的研究主要聚焦在数据划分和路由问题.相关研究成果主要有Exchange[57],Flux[56]等.MillWheel平台允许主节点根据工作节点的CPU、内存等资源使用情况自动对计算任务进行转移、拆分以及合并.文献[58-59]介绍了两种基于Storm平台的自适应动态负载调整策略.

减载(load shedding).由于早期硬件成本和集中式架构的限制,一些流处理系统会采用丢弃部分数据的方式来降低负载.文献[60-61]介绍了随机降载和语义降载的相关技术.由于分布式系统的设计理念即通过系统的可伸缩性应对负载的增加,因此大多数分布式流处理系统并没有把降载作为一个重要研究的问题.

2.6 系统容错

很多分布式系统运行在成百上千个相互协作的节点之上,即使单个节点出现问题的概率很低,也难

① 有关流处理中的事务并无明确定义,文献[51]中提出的统一事务模型可作参考

以保证每台机器都不会发生故障.一个好的分布式系统应具有良好的容错性(fault tolerant),即允许在部分节点出现错误时保证上层逻辑正常(或以一个稍低水平)运行.

一些无状态(stateless)的分布式流处理应用系统天生具有容错性.例如一个由双节点组成的温度检测报警系统,两个节点以抢占的方式(随机路由策略)从数据源获取实时产生的温度信息,并按照某预设值对越界温度报警.由于数据本身的持续性,当其中任一节点发生故障时,另一节点仍然可以保证报警业务的正常进行.S4 平台所提供的容错支持与之相似,所不同的是由于采用上下文敏感的路由策略,因此需要在备用节点启动新的任务处理实例以接替故障节点的工作.由于数据记录会随节点故障丢失,该方法只能保证最多一次处理语义.

现实情况中,绝大多数基于分布式流处理的应用都不是无状态的,简单的"进程接替"将难以满足要求.Hwang 等人针对上层应用需求,将故障恢复划分为精确恢复(precise recovery)、回滚恢复(rollback recovery)和有损回复(gap recovery)3 个级别[62].其中精确恢复是指系统恢复后的运行状态会和未发生故障的情况完全相同,唯一可能的影响是增加一些处理延迟;回滚恢复级别稍弱一些,它保证中间数据和状态不会丢失,虽然可能导致系统运行过程改变,但对运行结果不会产生影响(例如对一个幂等计算单元实例恢复导致部分数据重发);有损恢复是 3 种恢复级别中最弱的,它保证系统可以重新正确运行,但会丢失部分中间数据或状态.3 个恢复级别对 2.4 节所涉及的 3 类语义支持,如表 1 所示.由于回滚恢复从理论上可以支持所有语义保障,而其恢复的困难度又远远小于精确恢复,因此很多时候都会以它作为系统故障恢复的实现目标.

Table 1 Semantics Supported by Different Recovery Levels

表 1 不同恢复级别对处理语义的支持

Recovery Level	At Most Once	At Least Once	Exactly Once
Precise	Definitely	Definitely	Definitely
Rollback	Yes	Yes	Yes
Gap	Yes	No	No

现有故障恢复策略都是通过冗余备份的方式实现,常用方法可以分为 3 类,即主动备用(active standby)[63-64]、被动备用(passive standby)/上游备份(upstream backup)[15]和同步检查点(checkpoint)[16].其中主动备用和被动备用都是通过为正常工作的主节点(primary node)设置备用节点(backup node)实现.两者区别在于,主动备用策略中的备用节点会和主节点同时接收上游数据,它们以并行方式对相同的数据流进行处理并同时发往下游节点,下游节点需要按需对数据进行去重.当主节点故障时,备用节点可以无缝接替其工作,起到容错目的.被动备用策略中,备用节点不会从上游接收数据进行计算,它仅负责定期和主节点以异步方式进行状态同步(通常是增量同步).由于不能保证主节点发生故障时接替节点与其状态完全一致,因此被动备用策略还需要上游节点支持,即重发部分数据以追赶主节点的处理进度.一种比较极端的做法是不单独为处理过程中的节点设置定期的异步备份,仅通过上游节点的数据存储和重发实现容错,这种策略称为上游备份.和被动备用策略相比,它在降低运行时代价的同时进一步延长了故障恢复所需时间.同步检查点方式和被动备用类似,区别在于主节点和备用节点之间的数据传输改为同步策略,即数据在主节点被处理后,只有保证状态备份完成才会认为处理成功.虽然故障发生时省去了追赶进度的时间,但这种方式会显著增加处理延迟.

选用不同故障恢复手段的利益取舍如图 5 所示,其中 R_1 表示在保证低处理延迟的前提下,允许快速故障恢复的主动备用方法,但该方法至少会增加一倍的资源消耗,适合那些对处理过程和恢复时间要求极为苛刻的环境;R_2 代表同步检查点方法,它以牺牲处理延迟为代价,保证了较短的故障恢复时间,同时也不会造成过多的资源浪费,比较适合那些对实时性要求不高的流处理应用场景.R_3 代表被动备用(上游备份)方法,该方法所需的故障恢复时间最长,但同时拥有适当的资源消耗及处理延迟

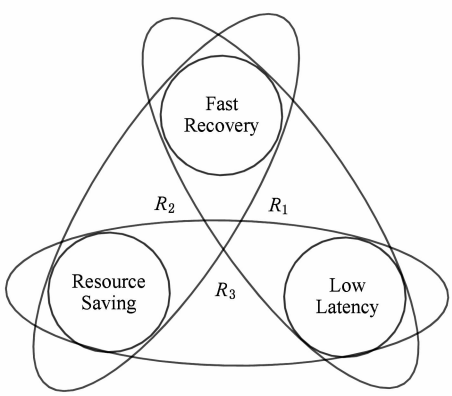


Fig. 5 Trade-offs between different recovery strategies.

图 5 不同恢复策略的利益取舍

影响,比较适合一些故障发生概率较低,且处理逻辑并不复杂的场景.由于故障恢复各级别的实现策略和计算逻辑息息相关[62],因此对于允许自定义计算单元的分布式流处理系统而言,只能通过一些诸如状态持久化、处理反馈等底层功能,对具体故障恢复工作提供一定支持.如图6所示,全局存储为被动备用提供了持久化的状态保存,分布式队列则可以天然支持上游备份.理想化的故障恢复策略应是根据每个计算单元的实际需求,为整个处理流程制定一个多种故障恢复方式相结合的解决方案.但面临和处理语义保障相同的问题,现有分布式流处理系统的开放特性为这种混合策略的制定带来了一定困难.

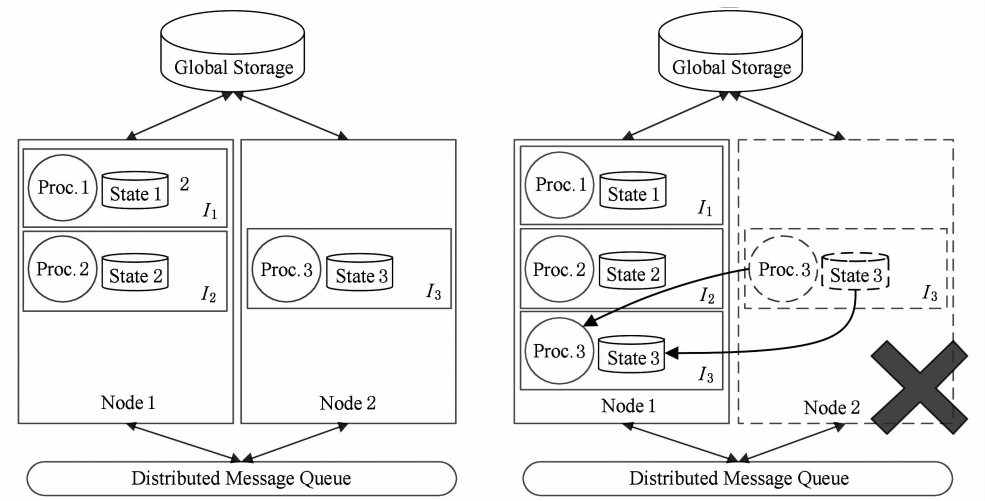


Fig. 6 Recovery supports of distributed systems.

图6 分布式系统对故障恢复的支持

2.7 其他问题

1) 高可用性(high availability).可用表示系统可以对外提供服务的一种状态,从用户角度看,如果在某个时间不能访问到系统则系统就被认为是不可用的[65].通常使用系统处于可用状态的时间与运行总时间之比来描述可用性.高可用性是一个比较模糊的概念,它需要二者比值达到某一较高数值.分布式流处理系统很多因素都可能对高可用造成影响,如存储设备的高延迟、过度频繁的负载均衡调度、语义保障失误造成系统崩溃、故障恢复时间过长等.因此为保障高可用性,需要考虑上述每一个具体设计问题可能导致的后果,杜绝"高可用性短板"的产生.

2) 算子及查询语言.文献[66]列举了实时流处理系统的8项需求,其中即包含查询语言支持.如第1节介绍,传统数据流管理系统,Aurora,STREAM,TelegraphCQ等都提供内置算子及在此之上的查询语言(通常是持续查询语言);近期提出的分布式流处理系统通常都是以平台形式存在,它们仅完成了一些较为底层的数据传输和任务控制等工作,需要用户在上层自定义计算单元,因而也就无法提供查询语言.现有系统中Spark Streaming由于使用了RDD,因此天生支持RDD上的转换操作.虽然在一些应用场景中表现良好,但这些操作本身的局限性依然导致系统难以变得通用.Storm的Trident提供了几个常用的操作算子,如连接(join)、聚合(aggregation)等.Squall[①]是由EPFL数据实验室开发的一个基于Storm的查询处理工具,它实现了选择(selection)、投影(projection)、去重(distinct)、聚合以及连结5种操作符,并提供类SQL语法的语言支持.然而该工具暂不能实时返回查询结果,也不支持嵌套查询,另外在语句丰富程度、查询优化和调度方面也存在很多不足.面对开放性带来的查询语言方面的挑战,一个可能的解决思路是参考传统SQL中的用户自定义函数(user defined function),首先抽象出部分算子,然后以用户自定义计算单元的方式提供扩展.

3 分布式流处理系统介绍

自2010年雅虎公司公开其通用分布式流处理平台S4起,许多用途相近又各具各色的平台相继被

① https://github.com/epfldata/squall

提出.这里我们选取 S4,Storm,Spark Streaming,Samza 和 MillWheel[①] 5 个较为有代表性的分布式流处理平台,下文将在对他们进行比较的同时(详见表 2),有侧重地对各平台的特点进行介绍.

Table 2 Comparisons of Distributed Stream Processing Systems

表 2 分布式流处理系统比较

Items	S4	Storm	Spark Streaming	Samza	MillWheel
Time	2010	2011	2012	2013	2013
Architecture	Decentralized	Weak centralized	Centralized	Centralized (with YARN)	Centralized
Data model	Event	Tuple	Object	Object	⟨key,val,ts⟩
Grain	Single record	Single record/batch	mini-batch	Single record	Single record/window
Data sorting	Not provide	Not provide	Not provide	Not provide	Provide
Routing	By key	Variety of/User defined	/	Depends on Kafka	By key
Load balancing	Static	Dynamic	/	Dynamic	Adaptive dynamic
Feedback	Not provide	Provide	/	Based on Kafka's ACK	Provide
Semantics	At most once	Exactly once	At least once	At least once	Exactly once
State storage	Local memory	Local memory/ Remote DB	Local memory	Embedded key-value store(with replication)	Backing store
Intermediate Result storage	Local memory	Local memory	Local memory/ Reliable File System	Kafka	Memory/ Backing store
Fault recovery	Stateless passive standby	Upstream backup	Checkpoint/ Parallel Recovery	Upstream backup	Checkpoint

3.1 S4

S4 是 Yahoo!公司于 2010 年开源的通用分布式流处理平台.它是上述 5 个平台中唯一一个采用去中心化结构的,各对等节点通过 ZooKeeper 进行协调工作.数据项在 S4 中被抽象为事件(event),流处理过程中每一阶段的计算单元会以多个 PE 的形式存在,每个 PE 只能处理同一 key 值的事件.由于缺少中心节点,S4 的设计遵循 actor 模式,即通过事件驱动对应处理 PE 的创建.虽然伸缩性和扩展性良好,但平台在数据路由、处理语义、故障恢复等方面存在很多局限和不足,例如只允许按 key 值对数据进行划分和路由[②],缺乏消息处理的反馈机制,无法进行有效的故障恢复等.

3.2 Storm

Storm 平台最早由 BackType 公司研发,后 Twitter 在 2011 年将其开源.相比 S4,Storm 最大的提升在于提供消息处理反馈机制和巧妙的利用异或计算保障记录被完全处理.平台采用弱中心化的结构,主节点只负责通过 ZooKeeper 向工作节点分配任务,不参与实际计算过程.同去中心化结构中每个节点都要掌握全局信息相比,这种模式大大降低了通信和同步代价,有利于在运行时进行任务调度.除提供一些初级的处理原语,Storm 的 Trident 还抽象出部分如连接、聚合等复杂运算的编程接口,并可以保证精确处理一次语义.虽然 Storm 平台在实际环境应用非常广泛,但依旧存在一定问题,例如很多特性需要在外部组件(如分布式队列等)的支持下由用户自行实现,Trident 的并行度存在一定瓶颈等.

3.3 Spark Streaming

Spark Streaming 是 Berkeley 在 2012 年开源的通用分布式流处理平台.作为伯克利数据分析堆栈(Berkeley data analytics stack, BDAS)的组成部分,Spark Streaming 和 Spark 数据处理引擎都使用了 RDD 作为其核心.与其他平台相比,Spark Streaming 最大的特点在于引入微批次的概念,将数据的处理粒度由单条记录粗化为数据集合,把对于数据流的操作看作是接连不断的批处理操作.类似于纯函数(pure function)的思想,绝大多数基于 RDD 的微批次变换操作都是明确的,即允许目标数据集在任何时间进行重建,不会受到外界因素干扰,这为并行故障恢复(parallel recovery)[16]带来了极大的方便.对于存在不确定因素的情况,该平台还允许以同步检查

① 除 MillWheel 外其余平台都已开源

② 0.6 版本

点的方式进行数据备份,确保对良好故障恢复的支持.就目前发展而言,Spark Streaming 还存在一些不足,例如增加了数据处理延迟、基于 RDD 的转换操作表达能力有限、不适用于更新外部状态(如全局索引)等.

3.4 Samza

Samza 是 LinkedIn 公司内部使用的分布式流处理平台,于 2013 年开源.该平台主要有两大特点:

1) 数据传输依赖于 LinkedIn 公司的另一开源项目——Kafka 分布式队列. Samza 平台的很多抽象方式,如数据流的分区(partition)、消费者(consumer)等都是 Kafka 中的概念.这种采用中间件的传输数据方式会对上游备份策略产生天然支持,能够轻松保障至少一次处理语义.

2) Samza 原生支持与 YARN[67] 协作,在后者的支持下可以与其他(非同类)系统共享计算节点,同时还可依靠 YARN 完成集群控制和故障恢复等工作.

Samza 提供了一个基于 LevelDB[68] 的内部键值存储用以存放状态,此外也支持可插拔式的存储实现,能够在被动备用策略支持下,较好完成故障恢复工作.该平台现有的不足在于暂不支持精确一次语义,数据处理的反馈受限于 Kafka 的顺序 ACK 机制等.

3.5 MillWheel

MillWheel 是 Google 于 2013 年公布的分布式流处理平台.同其他平台相比,MillWheel 面向的是带有时间戳的有序数据.它采用了低位线方式对数据进行批次切分和局部排序,以内部计时器触发机制保证数据按顺序处理.平台提供一个基于 BigTable 和 Spanner 的备份仓库,支持状态的持久化保存.在语义方面,MillWheel 利用 bloom-filter 和外部存储的去重提供精确一次处理保障,此外还支持不同级别的检查点策略.值得注意的是,MillWheel 是上述所有平台中唯一支持动态自适应负载均衡的一个,但正因如此,任务过重的计算节点可能会受负载均衡器的负面干预而无法正常工作. MillWheel 其他不足之处表现在,任务分配同样会受到 key 值有偏数据的影响以及当出现大量数据延迟时可能导致节点内存溢出.

4 分布式流处理相关应用

早期流数据处理技术主要应用于银行和股票交易领域,后逐渐延伸至地质、气象和天文观测等方面.本节主要针对分布式流处理的部分典型的应用领域,介绍其主要的研究问题和成果.

4.1 社交网络

以 Twitter 为代表的社交媒体已成为当下反映真实世界实时信息的巨大数据源,在这之上可以进行一系列有意义的挖掘,实时事件检测就是其中之一.社交媒体的流数据特性对在线的事件检测提出了新的要求:系统必须具有低响应延迟和高吞吐量;而社交流数据的规模之庞大以及事件检测自身的计算复杂性都使面向数据流的实时事件检测变得尤为困难.基于以上背景,已有一些人进行了相关研究. Petrovic 等人通过使用局部敏感哈希(LSH)技术提高了单机上基于数据流的实时事件检测的效率[69].然而,单机的吞吐量远无法满足处理大规模数据的要求.基于分布式实现事件检测又面临一大难题,即传统的数据流划分方法会降低事件检测的准确性. McCreadie 等人在文献[69]的基础上提出了一种针对大规模数据流进行分布式事件检测的方法,并在 Storm 平台上实现[70].他们保留了使用 LSH 产生键值的方法,将事件检测的过程分成了 4 个相对独立且易于实现并行处理的阶段.该方法相比于传统划分方法的优势在于处理大数据量时的高效性.

Lin 等人针对现有分布式数据流处理系统(如 S4 和 Storm)在错误恢复和全局数据存储方面的不足,对 S4 进行改进,提出 Pollux 系统,实现了对微博的精确实时检索[43].

诸如微信、手机游戏等支持移动社交的应用,通常会提供基于位置的服务,如寻找附近陌生人的功能.其中,用户被视为移动物体(moving objects),他们的位置信息构成数据流.之前针对移动物体数据的 KNN 查询研究都基于这样一个设置,即数据存储和 KNN 查询请求处理由同一个服务器完成. Yu 等人提出了一种方法,将对移动物体数据的实时 KNN 查询扩展到分布式系统上[71].他们设计了一种新的索引结构动态带状索引(dynamic strip index, DSI),并基于 DSI 提出了分布式的 KNN 检索算法 DKNN.

随着网站数据的急速增长,许多社交网站都会向用户推荐他可能感兴趣的内容(如 Twitter 推荐的 tweets、Flickr 推荐的图片、Youtube 推荐的视频等),以减少用户搜索的时间,这一功能由推荐系统完成.推荐系统所要处理的一个核心问题是如何在有新的内容发布之后,更新之前向每个用户推荐的

内容. Yang 等人针对实时推荐系统中 KNN 查询结果的更新问题,提出了一个更加高效的方法:为了避免重复计算,只对那些推荐结果集可能受到新发布内容影响的用户进行更新[72]. 为了快速识别这些用户,他们提出了一种新的索引结构 HDR-tree 以及它的变体 HDR*-tree.

对社交数据流的研究也延伸到了情感分析领域. Albert 和 Eibe 根据每一条 tweet 的形式信息实现了对单个用户的实时情绪分类[73];而 Wang 等人则选取了一个比较具体的问题研究环境[74],即 2012 年美国总统大选期间,公众情绪随着选举进行过程的变化,在 IBM 公司的分布式流处理平台 InfoSphere Streams① 上实现了实时的情绪变化报告.

4.2 网页广告

目前大多数广告推介都使用 behavioral targeting (BT)方法实现. 以往的 BT 是通过使用类似 Map-Reduce 的分布式框架在分布式文件系统上对数据进行聚类,但这种方法无法对临时数据进行分析. 针对此问题,Chandramouli 等人基于 TiMR 系统对网页广告推介进行了研究,提出一种在此系统上实现实时 BT 的方法[75].

用户与网站之间的交互(如关键字搜索、鼠标点击等)往往会形成一系列的事件流,对这些事件流进行关联分析对网站的广告商有着巨大意义. Google 在其广告系统中使用了 Photon[50],它以分布式文件系统 GFS 作为存储支撑,实现了对多个事件流的实时连接,并支持精确处理一次语义.

4.3 医学分析

医院 ICU 病房的病人通常需要配备许多不同种类的监测器,这些监测器会以各自的频率不断产生监测数据形成数据流,这些数据在监测器内存储一段时间之后就会被新采集的数据覆盖而丢失. Bar-Or 等人基于通用的流数据处理框架,提出了一种实时的分布式病理信号处理系统 BioStream[76]. BioStream 使用了额外的数据库存储监测数据,主要提供实时的病理信号分析和可视化两大功能,可以用于对病人的远程监控、警报和进一步的数据挖掘工作. 该系统还保证了实时响应和分布式环境下的数据一致性. Sow 等人使用分布式流处理平台 InfoSphere Streams 作为中间件,提出一个根据 ICU 病人的生理数据进行实时病情预测的系统[77]. 系统包括完整的数据读取、处理和预测结果输出的过程. 在此之前,他们还进行了基于病人的监测数据流进行病症分类的研究.

① http://www.ibm.com/software/products/zh/infosphere-streams

5 结束语

大数据时代,分布式流处理作为众多分布式数据处理模式中的一类,在面向动态数据的实时处理领域有着不可替代的地位. 本文回顾了该技术的发展历史,并详细分析了分布式流处理系统中相关设计问题. 第 3,4 节通过介绍现有分布式流处理系统和实际场景下的相关应用,展示了分布式流处理技术的重要影响. 通过比较分析我们发现,很多现有分布式流处理平台在计算模型、并行度控制、故障恢复、查询抽象及动态负载均衡方面还存在一些不足,而基于分布式流处理技术的数据分析及挖掘算法也还处于初级阶段. 综上,可从以下几方面开展研究工作:1)提出新颖合理的分布式流处理计算模型与体系架构;2)以可靠数据存储为支撑,针对分布式流处理系统中的并行读写进行优化;3)研究多个级别的混合式故障恢复策略;4)进行算子抽象及可扩展查询语言的设计;5)探究合理的自适应动态负载均衡机制;6)以实际应用为基础,提出基于分布式流处理的数据挖掘和分析算法.

参 考 文 献

[1] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[2] Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters [C] //Proc of the 6th Symp on Operating System Design and Implementation. San Francisco: USENIX Association, 2004: 137-150

[3] Schreier U, Pirahesh H, Agrawal R, et al. Alert: An architecture for transforming a passive DBMS into an active DBMS [C] //Proc of the 17th Int Conf on Very Large Data Bases. San Francisco, CA: Morgan Kaufmann, 1991: 469-478

[4] Kao B, Garcia-Molina H. An overview of real-time database systems [G] //Real Time Computing. Berlin: Springer, 1994: 261-282

[5] Paton N W, Dáz O. Active database systems [J]. ACM Computer Survey, 1999, 31(1): 63-103

[6] Belkin N J, Croft W B. Information filtering and information retrieval: Two sides of the same coin? [J]. Communications of the ACM, 1992, 35(12): 29-38

[7] Abadi D J, Carney D, Çetintemel U, et al. Aurora: A new model and architecture for data stream management [J]. VLDB Journal, 2003, 12(2): 120-139

[8] Chandrasekaran S, Cooper O, Deshpande A, et al. Telegraphcq: Continuous dataflow processing for an uncertain world [C/OL] //Proc of the 1st Biennial Conf on Innovative Data Systems Research. 2003 [2014-11-11]. http://cidrdb.org/2003Proceedings.zip

[9] Arasu A, Babcock B, Babu S, et al. Stream: The stanford data stream management system [OL]. 2004 [2014-11-11]. http://ilpubs.stanford.edu:8090/641/1/2004-20.pdf

[10] Cherniack M, Balakrishnan H, Balazinska M. Scalable distributed stream processing [C/OL] //Proc of the 1st Biennial Conf on Innovative Data Systems Research. 2003 [2014-11-11]. http://cidrdb.org/2003Proceedings.zip

[11] Shah M A, Hellerstein J M, Brewer E A. Highly-available, fault-tolerant, parallel dataflows [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2004: 827-838

[12] Abadi D J, Ahmad Y, Balazinska M, et al. The design of the borealis stream processing engine [C/OL] //Proc of the 2nd Biennial Conf on Innovative Data Systems Research. 2005: 277-289 [2014-11-11]. http://cidrdb.org/2005Proceedings.zip

[13] Apache Software Foundation. Welcome to Apache™ Hadoop® [EB/OL]. [2014-11-11]. http://hadoop.apache.org/

[14] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed stream computing platform [C] //Proc of the 10th IEEE Int Conf on Data Mining Workshops. Piscataway, NJ: IEEE, 2010: 170-177

[15] Toshniwal A, Taneja S, Shukla A, et al. Storm@twitter [C] //Proc of 2014 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2014: 147-156

[16] Zaharia M, Das T, Li H. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters [C] //Proc of the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'12). Berkeley, CA: USENIX Association, 2012

[17] Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2006: 407-418

[18] Agrawal J, Diao Y, Gyllstrom D, et al. Efficient pattern matching over event streams [C] //Proc of 2008 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2008: 147-160

[19] Demers A J, Gehrke J, Hong M, et al. Towards expressive publish/subscribe systems [C] //Proc of the 10th Int Conf on Extending Database Technology. Berlin: Springer, 2006: 627-644

[20] Demers A J, Gehrke J, Panda B, et al. Cayuga: A general purpose event monitoring system [C/OL] //Proc of the 3rd Biennial Conf on Innovative Data Systems Research. 2007: 412-422 [2014-11-11]. http://cidrdb.org/2007Proceedings.zip

[21] Brenna L, Gehrke J, Hong, M, et al. Distributed event stream processing with non-deterministic finite automata [C] //Proc of the 3rd ACM Int Conf on Distributed Event-Based Systems (DEBS 2009). New York: ACM, 2009

[22] Sun Dawei, Zhang Guangyan, Zheng Weimin. Big data stream computing: Technologies and instances [J]. Journal of Software, 2014, 25(4): 839-862 (in Chinese)
(孙大为, 张广艳, 郑纬民. 大数据流式计算: 关键技术及系统实例[J]. 软件学报, 2014, 25(4): 839-862)

[23] Li Guojie, Cheng Xueqi. Research status and scientific thinking of big data [J]. Bulletin of Chinese Academy of Sciences, 2012, 27(6): 647-657 (in Chinese)
(李国杰, 程学旗. 大数据研究: 未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考[J]. 中国科学院院刊, 2012, 27(6): 647-657)

[24] Meng Xiaofeng, Ci Xiang. Big data management: Concepts, techniques and challenges [J]. Journal of Computer Research and Development, 2013, 50(1): 146-169 (in Chinese)
(孟小峰, 慈祥. 大数据管理: 概念、技术与挑战[J]. 计算机研究与发展, 2013, 50(1): 146-169)

[25] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets [C] //Proc of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10. Berkeley, CA: USENIX Association, 2010

[26] Golab L, Özsu M T. Issues in data stream management [J]. SIGMOD Record, 2003, 32(2): 5-14

[27] Gehrke J, Korn F, Srivastava D. On computing correlated aggregates over continual data streams [C] //Proc of the 2001 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2001: 13-24

[28] Arasu A, Babu S, Widom J. The CQL continuous query language: Semantic foundations and query execution [J]. VLDB Journal, 2006, 15(2): 121-142

[29] Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems [C]//Proc of the 21st ACM SIGACT-SIGMOD-SIGART Symp on Principles of Database Systems. New York: ACM, 2002: 1-16

[30] Akidau T, Balikov A, Bekiroglu K, et al. Millwheel: Fault-tolerant stream processing at internet scale [J]. PVLDB, 2013, 6(11): 1033-1044

[31] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing [C] //Proc of the 9th USENIX Symp on Networked Systems Design and Implementation, NSDI 2012. Berkeley, CA: USENIX Association, 2012: 15-28

[32] Li J, Maier D, Tufte K, et al. Semantics and evaluation techniques for window aggregates in data streams [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2005: 311-322

[33] Li J, Tufte K, Shkapenyuk V, et al. Out-of-order processing: A new architecture for high-performance stream systems [J]. PVLDB, 2008, 1(1): 274-288

[34] Brito A, Fetzer C, Sturzrehm H, et al. Speculative out-of-order event processing with software transaction memory [C] //Proc of the 2nd Int Conf on Distributed Event-Based Systems, DEBS 2008. New York: ACM, 2008: 265-275

[35] Mutschler C, Philippsen M. Distributed low-latency out-of-order event processing for high data rate sensor streams [C] //Proc of the 27th IEEE Int Symp on Parallel and Distributed Processing (IPDPS 2013). Los Alamitos, CA: IEEE Computer Society, 2013: 1133-1144

[36] Stephens R. A survey of stream processing [J]. Acta Information, 1997, 34(7): 491-541

[37] Clinger D W. Foundations of actor semantics [OL]. [2014-11-11]. http://dspace.mit.edu/handle/1721.1/6935, 1981

[38] Hunt P, Konar M, Junqueira F P, et al. Zookeeper: Wait-free coordination for internet-scale systems [C] //Proc of USENIX Annual Technical Conf. Berkeley, CA: USENIX, 2010

[39] Wu S, Jiang D, Ooi B C, et al. Efficient b-tree based indexing for cloud data processing [J]. PVLDB, 2010, 3(1): 1207-1218

[40] Apache Software Foundation. Samza [EB/OL]. [2014-11-11]. http://samza.incubator.apache.org/

[41] Ongaro D, Rumble S M, Stutsman R, et al. Fast crash recovery in ramcloud [C] //Proc of the 23rd ACM Symp on Operating Systems Principles. New York: ACM, 2011: 29-41

[42] Lim H, Fan B, Andersen D, et al. SILT: A memory-efficient, high-performance key-value store [C] //Proc of the 23rd ACM Symp on Operating Systems Principles. New York: ACM, 2011: 1-13

[43] Lin Liwei, Yu Xiaohui, Koudas N. Pollux: Towards scalable distributed real-time search on microblogs [C] //Proc of the 16th Int Conf on Extending Database Technology. New York: ACM, 2013: 335-346

[44] UC Berkeley AMPLab. Tachyon Overview-Tachyon0.5.0 Documentation [EB/OL]. [2014-11-11]. http://tachyon-project.org/

[45] iMatix Corporation. Code connected-zeromq [EB/OL]. [2014-11-11]. http://zeromq.org/

[46] Lee Trustin. Netty Home [EB/OL]. [2014-11-11]. http://netty.io/

[47] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data [C] //Proc of the 7th Symp on Operating Systems Design and Implementation (OSDI'06). New York: ACM, 2006: 205-218

[48] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database [J]. ACM Trans on Computer Systems, 2013, 31(3): No.8

[49] Kreps J, Narkhede N, Rao J, et al. Kafka: A distributed messaging system for log processing [C] //Proc of the 6th NetDB. New York: ACM, 2011

[50] Ananthanarayanan R, Basker V, Das S, et al. Photon: Fault-tolerant and scalable joining of continuous data streams [C] //Proc of the 2013 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 577-588

[51] Botan I, Fischer P M, Kossmann D, et al. Transactional stream processing [C] //Proc of the 15th Int Conf on Extending Database Technology. New York: ACM, 2012: 204-215

[52] Wei Hong, Stonebraker M. Optimization of parallel query execution plans in XPRS [C] //Proc of the 1st Int Conf on Parallel and Distributed Information Systems. Berlin: Springer, 1991: 218-225

[53] Dayarathna M, Suzumura T. Hirundo: A mechanism for automated production of optimized data stream graphs [C] //Proc of the 3rd ACM/SPEC Int Conf on Performance Engineering. New York: ACM, 2012: 335-346

[54] Babu S, Widom J. Streamon: An adaptive engine for stream query processing [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2004: 931-932

[55] Avnur R, Hellerstein J M. Eddies: Continuously adaptive query processing [C] //Proc of the 2000 ACM SIGMOD Int Conference on Management of data. New York: ACM, 2000: 261-272

[56] Shah M A, Hellerstein J M, Chandrasekaran S. Franklin, et al. Flux: An adaptive partitioning operator for continuous query systems [C] //Proc of the 19th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2003: 25-36

[57] Graefe G. Encapsulation of parallelism in the volcano query processing system [C] //Proc of the 1990 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 1990: 102-111

[58] Aniello L, Baldoni R, Querzoni L. Adaptive online scheduling in storm [C] //Proc of the 7th ACM Int Conf on Distributed Event-Based Systems. New York: ACM, 2013: 207-218

[59] Sax M J, Castellanos M, Chen Q, et al. Aeolus: An optimizer for distributed intra-node-parallel streaming systems [C] //Proc of the 29th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2013: 1280-1283

[60] Das A, Gehrke J, Riedewald M. Approximate join processing over data streams [C] //Proc of the 2003 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2003: 40-51

[61] Ayad A, Naughton J F. Static optimization of conjunctive queries with sliding windows over infinite streams [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2004: 419-430

[62] Hwang J, Balazinska M, Rasin A, et al. High-availability algorithms for distributed stream processing [C] //Proc of the 21st Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2005: 779-790

[63] Balazinska M, Balakrishnan H, Madden S, et al. Fault-tolerance in the borealis distributed stream processing system [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2005: 13-24

[64] Hwang J, Çetintemel U, Zdonik S B. Fast and highly-available stream processing over wide area networks [C] //Proc of the 24th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2008: 804-813

[65] Piedad F, Hawkins M. High Availability: Design, Techniques, and Processes [M]. London: Prentice Hall Professional, 2001

[66] Stonebraker M, Çetintemel U, Zdonik S B. The 8 requirements of real-time stream processing [J]. SIGMOD Record, 2005, 34(4): 42-47

[67] Apache Software Foundation. Yarn [EB/OL]. [2014-11-11]. http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/

[68] Google. Leveldb [EB/OL]. [2014-11-11]. http://leveldb.org/

[69] Petrovic S, Osborne M, Lavrenko V. Streaming first story detection with application to twitter [C] //Proc In Human Language Technologies: Conf of the North American Chapter of the Association of Computational Linguistics. New York: ACM, 2010: 181-189

[70] McCreadie R, Macdonald C, Ounis I, et al. Scalable distributed event detection for twitter [C] //Proc of the 2013 IEEE Int Conf on Big Data. Piscataway, NJ: IEEE, 2013: 543-549

[71] Yu Ziqiang, Liu Yang, Yu Xiaohui, et al. Scalable distributed processing of k nearest neighbor queries over moving objects [J]. IEEE Trans on Knowledge and Data Engineering, 2014, PP(99): 1

[72] Yang Chong, Yu Xiaohui, Liu Yang. Continuous knn join processing for real-time recommendation [C] //Proc of the 2014 Int Conf on Data Mining. Berlin: Springer, 2014

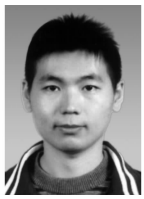
[73] Bifet A, Frank E. Sentiment knowledge discovery in twitter streaming data [C] //Proc of the 13th Int Conf on Discovery Science. Berlin: Springer, 2010: 1-15

[74] Wang Hao, Can D, Kazemzadeh A, et al. A system for real-time twitter sentiment analysis of 2012 us presidential election cycle [C] //Proc of 2012 ACL System Demonstrations. New York: ACM, 2012: 115-120

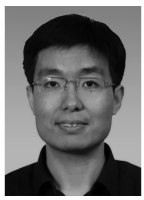
[75] Chandramouli B, Goldstein J, Duan S. Temporal analytics on big data for web advertising [C] //Proc of the 28th IEEE Int Conf on Data Engineering (ICDE 2012). Los Alamitos, CA: IEEE, 2012: 90-101

[76] Bar-Or A, Healey J, Kontothanassis L, et al. Biostream: A system architecture for real-time processing of physiological signals [C] //Proc of teh 26th Annual Conf of the IEMBS'04. Piscataway, NJ: IEEE, 2004: 3101-3104

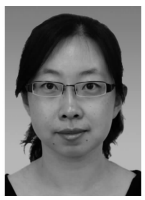
[77] Sow D, Biem A, Sun J, et al. Real-time prognosis of icu physiological data streams [C] //Proc of 2010 Annual Int Conf on Engineering in Medcine and Biology Society. Piscataway, NJ: IEEE, 2010: 6785-6788



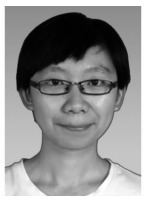
Cui Xingcan, born in 1989. PhD candidate. Student member of China Computer Federation. His research interests include distributed data processing and data quality management.



Yu Xiaohui, born in 1977. Professor and PhD supervisor. Member of China Computer Federation. His main research interests include data management and data mining(xyu@sdu.edu.cn).



Liu Yang, born in 1977. Associate professor. Member of China Computer Federation. Her main research interests include data mining and data management.



Lü Zhaoyang, born in 1992. Master candidate. Her main research interest is stream processing(zhylv@mail.sdu.edu.cn).